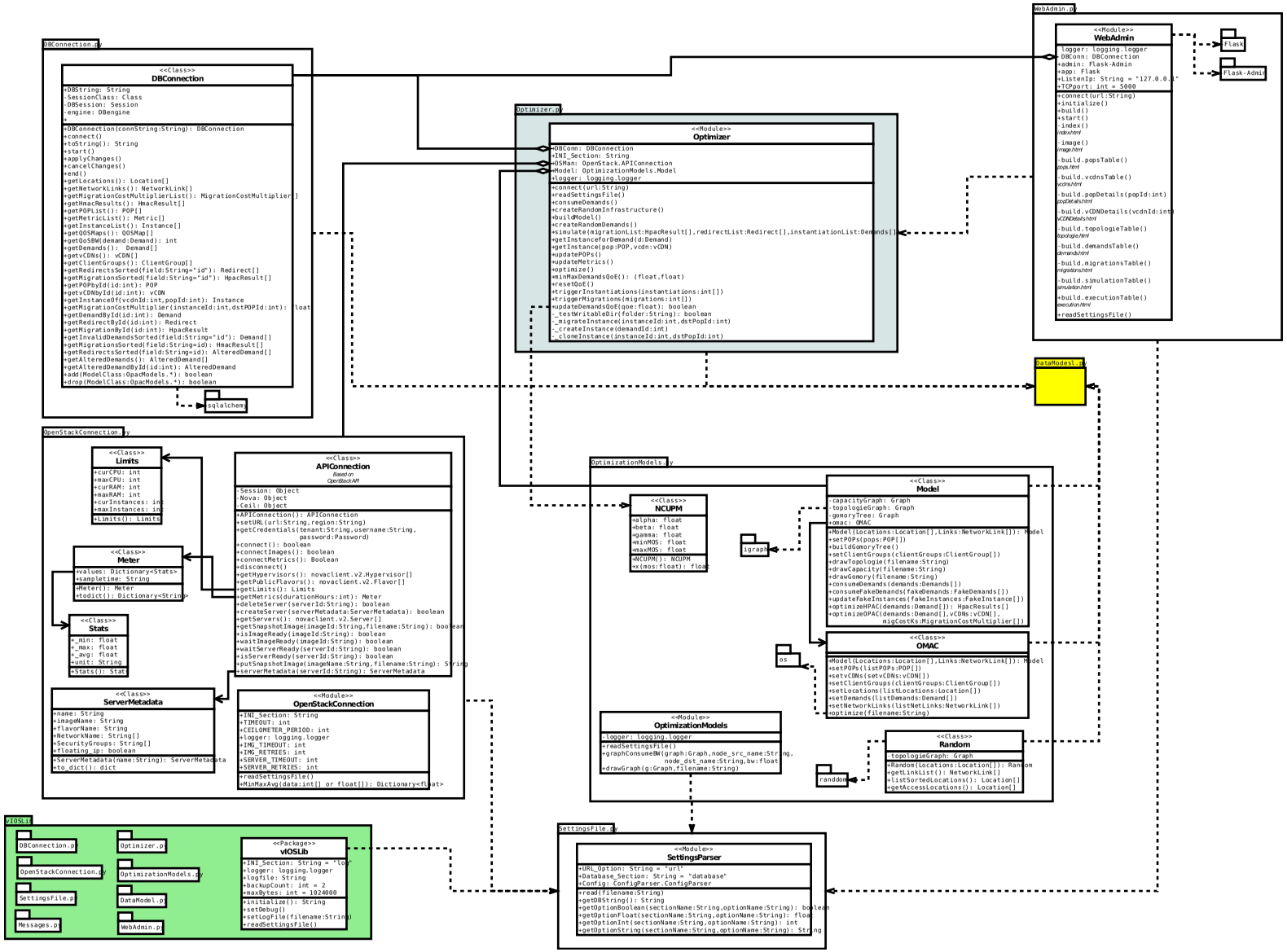
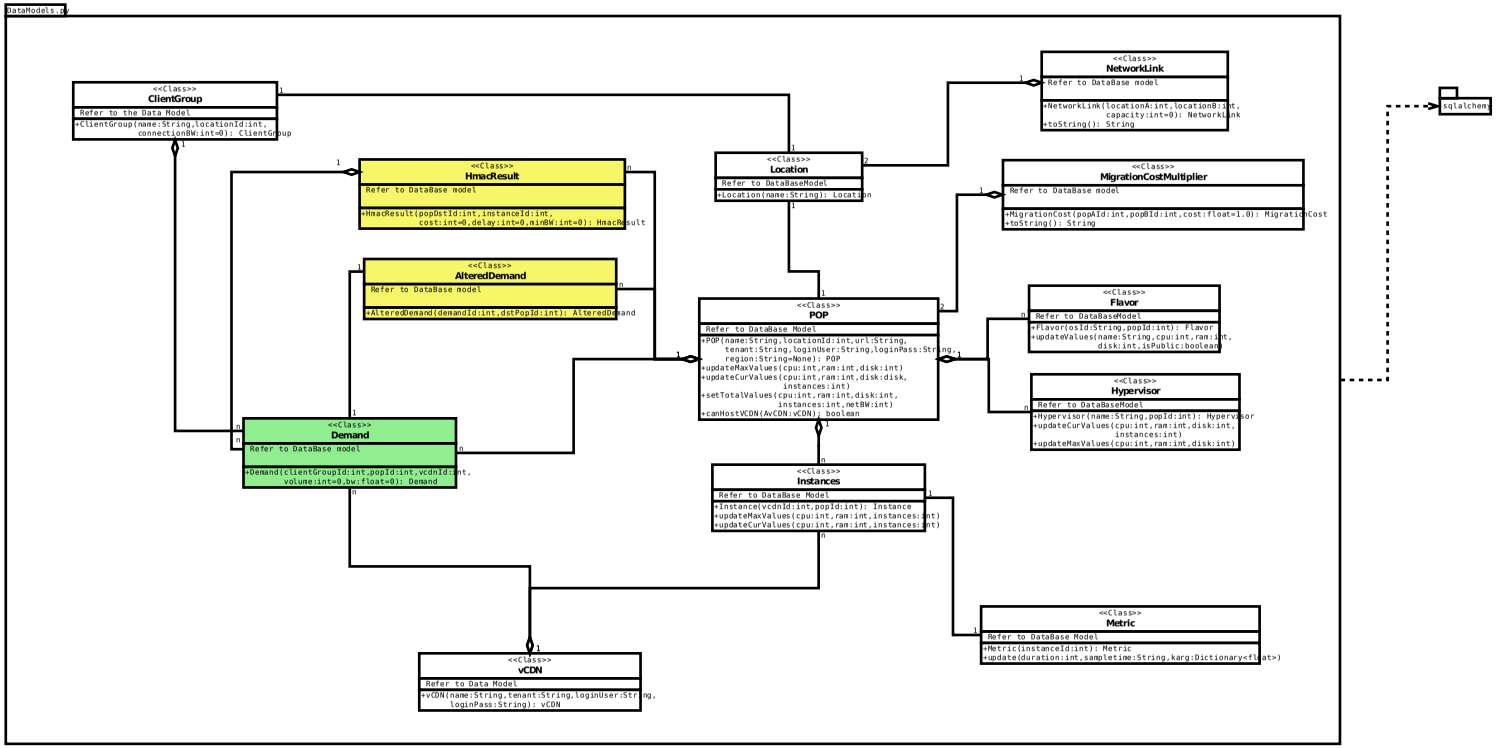

DBConnection.py
<<Class>>
DBConnection
+DBString: String
-SessionClass: Class
-DBSession: Session
-engine: DBengine
+DBConnection(connString:String): DBConnection
+connect()
+toString(): String
+start()
+applyChanges()
+cancelChanges()
+end()
+getLocations(): Location[]
+getNetworkLinks(): NetworkLink[]
+getMigrationCostMultiplierList(): MigrationCostMultiplier[]
+getHmacResults(): HmacResult[]
+getPOPList(): POP[]
+getMetricList(): Metric[]
+getInstanceList(): Instance[]
+getOOSMaps(): OOSMap[]
+getQoSBW(demand:Demand): int
+getDemands(): Demand[]
+getvCDNs(): vCDN[]
+getClientGroups(): ClientGroup[]
+getRedirectsSorted(field:String="id"): Redirect[]
+getMigrationsSorted(field:String="id"): HpacResult[]
+getPOPbyId(id:int): POP
+getvCDNbyId(id:int): vCDN
+getInstanceOf(vcdnId:int,popId:int): Instance
+getMigrationCostMultiplier(instanceId:int,dstPOPId:int): float
+getDemandById(id:int): Demand
+getRedirectById(id:int): Redirect
+getMigrationById(id:int): HpacResult
+getInvalidDemandsSorted(field:String="id"): Demand[]
+getMigrationsSorted(field:String=id): HmacResult[]
+getRedirectsSorted(field:String=id): AlteredDemand[]
+getAlteredDemands(): AlteredDemand[]
+getAlteredDemandById(id:int): AlteredDemand
+add(ModelClass:OpacModels.*): boolean
+drop(ModelClass:OpacModels.*): boolean
sqlalchemy
Optimizer.py
<<Module>>
Optimizer
+DBConn: DBConnection
+INI_Section: String
+OSMan: OpenStack.APIConnection
+Model: OptimizationModels.Model
+logger: logging.logger
+connect(url:String)
+readSettingsFile()
+consumeDemands()
+createRandomInfrastructure()
+buildModel()
+createRandomDemands()
+simulate(migrationList:HpacResult[],redirectList:Redirect[],instantiationList:Demands[])
+getInstanceforDemand(d:Demand)
+getInstance(pop:POP,vcdn:vCDN)
+updatePOPs()
+updateMetrics()
+optimize()
+minMaxDemandsQoE(): (float,float)
+resetQoE()
+triggerInstantiations(instantiations:int[])
+triggerMigrations(migrations:int[])
+updateDemandsQoE(qoe:float): boolean
-_testWritableDir(folder:String): boolean
-_migrateInstance(instanceId:int,dstPopId:int)
-_createInstance(demandId:int)
-_cloneInstance(instanceId:int,dstPopId:int)
WebAdmin.py
<<Module>>
WebAdmin
-logger: logging.logger
-DBConn: DBConnection
+admin: Flask-Admin
+app: Flask
+ListenIp: String = "127.0.0.1"
+TCPport: int = 5000
+connect(url:String)
+initialize()
+build()
+start()
-index()
/index.html
-image()
/image.html
-build.popsTable()
/pops.html
-build.vcdnsTable()
/vcdns.html
-build.popDetails(popId:int)
/popDetails.html
-build.vCDNDetails(vcdnId:int)
/vCDNDetails.html
-build.topologieTable()
/topologie.html
-build.demandsTable()
/demands.html
-build.migrationsTable()
/migrations.html
-build.simulationTable()
/simulation.html
+build.executionTable()
/execution.html
+readSettingsFile()
Flask
Flask-Admin
DataModel.py
OpenStackConnection.py
<<Class>>
Limits
+curCPU: int
+maxCPU: int
+curRAM: int
+maxRAM: int
+curInstances: int
+maxInstances: int
+Limits(): Limits
<<Class>>
Meter
+values: Dictionary<Stats>
+sampletime: String
+Meter(): Meter
+todict(): Dictionary<String>
<<Class>>
Stats
+_min: float
+_max: float
+_avg: float
+unit: String
+Stats(): Stat
<<Class>>
APIConnection
Based on OpenStackAPI
-Session: Object
-Nova: Object
-Ceil: Object
+APIConnection(): APIConnection
+setURL(url:String,region:String)
+getCredentials(tenant:String,username:String, password:Password)
+connect(): boolean
+connectImages(): boolean
+connectMetrics(): Boolean
+disconnect()
+getHypervisors(): novaclient.v2.Hypervisor[]
+getPublicFlavors(): novaclient.v2.Flavor[]
+getLimits(): Limits
+getMetrics(durationHours:int): Meter
+deleteServer(serverId:String): boolean
+createServer(serverMetadata:ServerMetadata): boolean
+getServers(): novaclient.v2.Server[]
+getSnapshotImage(imageId:String,filename:String): boolean
+isImageReady(imageId:String): boolean
+waitImageReady(imageId:String): boolean
+waitServerReady(serverId:String): boolean
+isServerReady(serverId:String): boolean
+putSnapshotImage(imageName:String,filename:String): String
+serverMetadata(serverId:String): ServerMetadata
<<Class>>
ServerMetadata
+name: String
+imageName: String
+flavorName: String
+NetworkName: String[]
+SecurityGroups: String[]
+floating_ip: boolean
+ServerMetadata(name:String): ServerMetadata
+to_dict(): dict
<<Module>>
OpenStackConnection
+INI_Section: String
+TIMEOUT: int
+CEILOMETER_PERIOD: int
+logger: logging.logger
+IMG_TIMEOUT: int
+IMG_RETRIES: int
+SERVER_TIMEOUT: int
+SERVER_RETRIES: int
+readSettingsFile()
+MinMaxAvg(data:int[] or float[]): Dictionary<float>
OptimizationModels.py
<<Class>>
NCUPM
+alpha: float
+beta: float
+gamma: float
+minMOS: float
+maxMOS: float
+NCUPM(): NCUPM
+x(nos:float): float
igraph
<<Class>>
Model
-capacityGraph: Graph
-topologieGraph: Graph
-gomoryTree: Graph
-omac: OMAC
+Model(Locations:Location[],Links:NetworkLink[]): Model
+setPOPs(pops:POP[])
+buildGomoryTree()
+setClientGroups(clientGroups:ClientGroup[])
+drawTopologie(filename:String)
+drawCapacity(filename:String)
+drawGomory(filename:String)
+consumeDemands(demands:Demands[])
+consumeFakeDemands(fakeDemands:FakeDemands[])
+updateFakeInstances(fakeInstances:FakeInstance[])
+optimizeHPAC(demands:Demand[]): HpacResults[]
+optimizeOPAC(demands:Demand[],vCDNs:vCDN[], migCostKs:MigrationCostMultiplier[])
os
<<Class>>
OMAC
+Model(Locations:Location[],Links:NetworkLink[]): Model
+setPOPs(listPOPs:POP[])
+setvCDNs(setvCDNs:vCDN[])
+setClientGroups(clientGroups:ClientGroup[])
+setLocations(listLocations:Location[])
+setDemands(listDemands:Demand[])
+setNetworkLinks(listNetLinks:NetworkLink[])
+optimize(filename:String)
<<Module>>
OptimizationModels
-logger: logging.logger
+readSettingsFile()
+graphConsumeBW(graph:Graph,node_src_name:String, node_dst_name:String,bw:float)
+drawGraph(g:Graph,filename:String)
<<Class>>
Random
-topologieGraph: Graph
+Random(Locations:Location[]): Random
+getLinkList(): NetworkLink[]
+listSortedLocations(): Location[]
+getAccessLocations(): Location[]
randdom
vIOSLit
DBConnection.py
Optimizer.py
OpenStackConnection.py
OptimizationModels.py
SettingsFile.py
DataModel.py
Messages.py
WebAdmin.py
<<Package>>
vIOSLib
+INI_Section: String = "log"
+logger: logging.logger
+logfile: String
+backupCount: int = 2
+maxBytes: int = 1024000
+initialize(): String
+setDebug()
+setLogFile(filename:String)
+readSettingsFile()
SettingsFile.py
<<Module>>
SettingsParser
+URL_Option: String = "url"
+Database_Section: String = "database"
+Config: ConfigParser.ConfigParser
+read(filename:String)
+getDBString(): String
+getOptionBoolean(sectionName:String,optionName:String): boolean
+getOptionFloat(sectionName:String,optionName:String): float
+getOptionInt(sectionName:String,optionName:String): int
+getOptionString(sectionName:String,optionName:String): String
Class Diagram
by Ignacio Tamayo
vIOSimulator - Télécom SudParis
August 2016
Ver: 1.4



Class Diagram
by Ignacio Tamayo
vIOSimulator - Télécom SudParis
August 2016
Ver: 1.4